

# Northumbria Research Link

Citation: Algabasi, Abdullatif and Maheri, Alireza (2015) A genetic algorithm for solving classical job shop scheduling problems. In: International Conference on Advances in Mechanical Engineering (ICAME'15), 13-15 May 2015, Istanbul.

URL:

This version was downloaded from Northumbria Research Link:  
<http://nrl.northumbria.ac.uk/id/eprint/22520/>

Northumbria University has developed Northumbria Research Link (NRL) to enable users to access the University's research output. Copyright © and moral rights for items on NRL are retained by the individual author(s) and/or other copyright owners. Single copies of full items can be reproduced, displayed or performed, and given to third parties in any format or medium for personal research or study, educational, or not-for-profit purposes without prior permission or charge, provided the authors, title and full bibliographic details are given, as well as a hyperlink and/or URL to the original metadata page. The content must not be changed in any way. Full items must not be sold commercially in any format or medium without formal permission of the copyright holder. The full policy is available online: <http://nrl.northumbria.ac.uk/policies.html>

This document may differ from the final, published version of the research and has been made available online in accordance with publisher policies. To read and/or cite from the published version of the research, please visit the publisher's website (a subscription may be required.)



**Northumbria  
University**  
NEWCASTLE



**UniversityLibrary**

## A GENETIC ALGORITHM FOR SOLVING CLASSICAL JOB SHOP SCHEDULING PROBLEMS

**\*Alireza Maheri**  
Northumbria University  
Newcastle upon Tyne, UK

**Abdullatif Algabasi**  
Northumbria University  
Newcastle upon Tyne, UK

*Keywords: classical job shop scheduling, generalised job shop scheduling, JSS, JSSP, genetic algorithm*

*\* Corresponding author:., Phone: +44 (0) 191 227 3860*

*E-mail address: Alireza.Maheri@northumbria.ac.uk*

### ABSTRACT

This paper presents a new genetic algorithm (GA) for solving job shop scheduling (JSS) problem, in which a two-dimensional chromosome is used to represent individuals. Classical JSS problem is formulated in a more realistic and general way in which the expected arrival time for each job and machine availability for each machine are also taken into account. This formulation also allows solving special cases of returning tasks, in which a job can have more than one task allocated for a specific machine. Through a case study, the performance of the developed GA in solving generalised classic JSS problem is evaluated and presented.

### INTRODUCTION

Job shop scheduling (JSS) problem is a class of optimisation problem with wide range of application in manufacturing. Along with advances in manufacturing and automation, JSS has received significant attention from research community during the past three decades (e.g. see [1] through [22]).

JSS problem can be classified as classical and flexible. In classical JSS (CJSS) problems, machines required to deliver all tasks are of different types. For a total number of  $N_t$  tasks to be delivered, there are  $N_m$  machines of  $N_m$  different types. In other words, tasks are assigned to specific machines. In flexible JSS (FJSS), there are  $N_m$  identical machines and all  $N_t$  tasks can be processed on any of the available machines.

JSS can be also classified into static (SJSS) and dynamic (DJSS) in terms of the available information prior to commencing solving the problem. In SJSS all jobs are known and are fixed during the process time. In DJSS, some jobs or some information about some jobs is not available at the

beginning of the process. A DJSS changes the processing schedule as new information become available or new jobs are added to the process.

Solutions to a JSS optimisation problem, irrespective of the type of the problem, are evaluated with respect to a series of criteria. The most common assessment criterion is the overall processing time (makespan).

A JSS problem includes a number of jobs, each made of a number of tasks that need to be processed following a predefined precedence order. There are some essential assumptions in defining a CJSS problem:

- More than one task cannot be processed on the same machine at the same time
- Each task has a fix processing time on its corresponding machine
- A process cannot be interrupted before completion.

A CJSS problem as defined above can be used to model a vast number of real-life applications. However, referring to the literature, one notices that many reported formulations apply some simple and unnecessary constraints to the CJSS problem. These constraints as detailed in the next section, make the CJSS problem less realistic with less practical use. The aim of this paper is to reformulate a CJSS as realistic as possible and develop a genetic algorithm for solving CJSS.

### CLASSICAL JSS: MORE REALISTIC FORMULATION

Taking the overall makespan as the objective of optimisation, the CJSS single objective optimisation problem is normally formulated as:

$$\min \text{makespan} = \max \{C_{n_i, j, j}\}, \forall j \in \{1, 2, \dots, N_j\} \quad (1)$$

subject to:

$$S_{i+1,j} \geq C_{i,j} = S_{i,j} + T_{i,j}, \forall i \in \{1, 2, \dots, n_{t,j}\}, \forall j \in \{1, 2, \dots, N_j\} \quad (2)$$

$$M_{i,j} = M_{k,l} \Rightarrow S_{i,j} \geq S_{k,l} + T_{k,l} \vee S_{k,l} \geq S_{i,j} + T_{i,j}, \\ \forall i \in \{1, 2, \dots, n_{t,j}\}, \forall k \in \{1, 2, \dots, n_{t,l}\} \quad (3)$$

where,  $N_j$  is the number of jobs,  $S_{i,j}$ ,  $T_{i,j}$  and  $C_{i,j}$  are the start time, process time and completion time of task  $t_{i,j}$ ,  $M_{i,j}$  is the machine associated to task  $t_{i,j}$ ,  $n_{t,j}$  is the number of tasks on job  $j$  and  $C_{n_{t,j},j}$  is the completion time of the last task of job  $j$ .

Constraint (2) guarantees the precedence of tasks on each job and that a process cannot be interrupted before completion. Constraint (3) denotes that two tasks are not processed on the same machine at the same time. If two tasks are assigned to the same machine, the start time of one should be after the completion time of the other.

Besides the essential constraints above, there are some constraints, which are not explicitly mentioned in the formulation but are implied in most of CJSS case studies and benchmark problems reported in the literature. These constraints reduce the potentials of formulated CJSS in solving real-life problem. These constraints are as follows:

$$S_{1,j} \geq 0, \forall j \in \{1, 2, \dots, N_j\} \quad (4)$$

Constraint (4) implies that all jobs (the first task of each job) can be started at time  $t=0$ . In real-life practice, this is not always the case. In many manufacturing lines, jobs become ready to start at different times, depending on their arrival time from other parts of the production line. This also implies that, if the availability of machines allows, there is always enough workforce for setting up the first task of all jobs simultaneously. This is not the case for many workshops and manufacturing lines. The second implicit constraint is given by

$$MA_m = [0, \infty), \forall m \in \{1, 2, \dots, N_m\} \quad (5)$$

where,  $MA_m$  stands for the availability period of machine  $m$ ,  $N_m$  is the number of machines required to deliver all jobs. Constraint (5) implies that the availability of all machines is not limited by any other constraints. Implementing this constraint leads to exclusion of some realistic cases, such as unavailability of machine due to repair, from the formulation.

In addition to the implicit constraints (4) and (5), the form of CJSS benchmark problems in the literature imposes three additional constraints, making these benchmark problems even

more deviated from realistic cases. These constraints are as follow:

$$M_{i,j} \neq M_{k,l}, \forall i \neq k, \quad (6)$$

$$n_{m,j} = \text{const}, \forall j \in \{1, 2, \dots, N_j\}, \text{ and} \quad (7)$$

$$n_{t,j} = \text{const}, \forall j \in \{1, 2, \dots, N_j\} \quad (8)$$

where,  $n_{m,j}$  stands for the number of machines needed to deliver job  $j$ . Constraints (6) through (8), respectively, denote that no two tasks from the same job are assigned to the same machine, that the number of machines required for each job is the same and that all machines are used for all jobs. None of these constraints is affiliated to most of real life CJSS problems.

The proposed formulation of CJSS problem is more general and allows us to solve more realistic cases. In the new formulation:

- each job  $j$  has an arrival time  $JA_j$ ,
- the availability of machines can be limited (for example for repair or engagement in other sets of jobs),
- the number of tasks on different jobs can be different,
- different jobs can use different number of machines,
- each job can have more than one task on one machine (for example in case of the manufacturing of crankshaft, in which the work piece needs to visit the CNC machine twice at different stages before and after heat treatment).

In this formulation, Constraints (4) and (5) are modified and Constraints (6) to (8) are removed. The optimisation problem is transformed to:

$$\min \text{makespan} = \max\{C_{n_{t,j},j}\}, \forall j \in \{1, 2, \dots, N_j\} \quad (1\text{-repeat})$$

subject to:

$$S_{i+1,j} \geq C_{i,j} = S_{i,j} + T_{i,j}, \forall i \in \{1, 2, \dots, n_{t,j}\}, \forall j \in \{1, 2, \dots, N_j\} \quad (2\text{-repeat})$$

$$M_{i,j} = M_{k,l} \Rightarrow S_{i,j} \geq S_{k,l} + T_{k,l} \vee S_{k,l} \geq S_{i,j} + T_{i,j}, \\ \forall i \in \{1, 2, \dots, n_{t,j}\}, \forall k \in \{1, 2, \dots, n_{t,l}\} \quad (3\text{-repeat})$$

$$S_{1,j} \geq JA_j, \forall j \in \{1, 2, \dots, N_j\} \quad (9)$$

$$MA_m \subseteq [0, \infty), \forall m \in \{1, 2, \dots, N_m\} \quad (10)$$

in which,  $JA_j$  is the arrival time of job  $j$ .

Figure 1 shows how a typical CJSS problem can be defined according to formulation above (coded in MATLAB). Parameter  $job(j).task$  is a vector of length  $n_{t,j}$  containing task names (or numbers) in job  $j$  with the right precedence;  $job(j).machine$  and  $job(j).time$  are also vectors of length  $n_{t,j}$  containing the name (or number) of machines associated to the tasks in  $job(j).task$  and the processing time for each task.  $machine\_av$  is a matrix of size  $2 \times N_m$  containing the availability intervals for each machine. As it can be observed from this example, the seven jobs on this problem contain different number of tasks. Jobs 1 and 7 have arrival time of  $t = 50$  units, while the other jobs can be started at time  $t = 0$ . In case of job 1, it can be seen that both tasks 4 and 1 are processed on the same machine (machine 1). That is, the number of tasks on this job (4) is different from the number of required machines (3). Machines 3, 4, 5 and 6 have limited availability.

```

job(1).task=[4,9,1,6];
job(1).machine=[1,2,1,4];
job(1).time=[20,28,18,13];
job(1).arrival_time=[50];

job(2).task=[2,5,7];
job(2).machine=[2,5,3];
job(2).time=[16,17,15];
job(2).arrival_time=[0];

job(3).task=[3,8,10,100];
job(3).machine=[4,1,3,5];
job(3).time=[14,22,24,26];
job(3).arrival_time=[0];

job(4).task=[103,108,110,111,112];
job(4).machine=[1,7,3,5,4];
job(4).time=[35,30,25,30,15];
job(4).arrival_time=[0];

job(5).task=[203,208,210,211,212];
job(5).machine=[1,9,3,5,4];
job(5).time=[15,10,15,20,15];
job(5).arrival_time=[0];

job(6).task=[22,25,27];
job(6).machine=[2,5,3];
job(6).time=[16,17,15];
job(6).arrival_time=[0];

job(7).task=[301,302,404,304,305,311,307,288,309,310];
job(7).machine=[7,9,3,5,4,2,9,5,6,7];
job(7).time=[15,10,15,20,15,20,20,25,25];
job(7).arrival_time=[50];

machine_av=[0,0,10,20,0,5,0,0,0,0;
            Inf,Inf,300,250,400,Inf,Inf,Inf,Inf,Inf];

```

**FIGURE 1-A TYPICAL REALISTIC CJSS PROBLEM**

## GENETIC ALGORITHM

The GA developed for this problem is different from traditional GA in terms of chromosome definition. The

chromosome is a matrix of size  $N_M \times n_{t,m,\max}$ , where  $n_{t,m,\max}$  is the maximum number of tasks on each machine given by Equation 11. It should be noted that in generalised CJSS formulation, the number of tasks to be delivered by different machines  $n_{t,m}$  can be different.

$$n_{t,m,\max} = \max\{n_{t,m}, \forall m \in \{1, 2, \dots, N_m\}\} \quad (11)$$

Each row of this matrix presents the set of tasks which are allocated for the associated machine to that row. Figure 2 shows a typical chromosome for the problem of Figure 1. In this chromosome each gene refers to a task number (or name). Genes with values 0 are blank tasks. Figure A1 of the appendix shows the Gantt chart for this solution.

Machine 1	203	103	8	4	1	0	0
Machine 2	2	22	9	311	0	0	0
Machine 3	210	27	7	110	10	404	0
Machine 4	3	112	212	6	305	0	0
Machine 5	25	5	111	211	100	304	288
Machine 6	309	0	0	0	0	0	0
Machine 7	301	108	310	0	0	0	0
Machine 9	208	302	307	0	0	0	0

**FIGURE 2-A TYPICAL CHROMOSOME**

In the proposed GA a geometric crossover is used. A randomly located horizontal cut line splits the parent chromosomes into upper and lower parts. Two children are made as the result of each crossover by swapping the upper and lower parts. Mutation operator is applied to a single machine by reordering tasks on that machine. In a randomly selected row (machine), two randomly selected genes (tasks) are swapped resulting to reordering tasks on that machine.

## RESULTS AND DISCUSSION

To evaluate the performance of the developed GA, it was used to optimise the makespan for a series of benchmark problems with known optimum solutions, including LA01 through LA10, LA15, LA 20, LA 25, LA30, LA35 and LA40. In all cases the developed GA was able to find the optimum solution in all its runs. Figure A2 shows the Gantt charts for jobs and machines of the benchmark LA05, as if there was no precedence constraints. In other words, these charts can be used to identify the shortest possible makespan as if there was no constraint on the task precedence. For this benchmark problem, the minimum (ideal) machine completion times are, respectively, 463, 532, 391, 304 and 593 units leading to an ideal total makespan of 593 units. Figure A3 shows the optimum solution found by the present GA for this problem with optimum machine completion times of 567, 532, 460, 589 and 593, making an optimum makespan of 593 units.

The developed GA was also used to solve the problem of Figure 1. The Gantt charts of unconstrained solution are shown in Figure A4. The unconstrained completion times for jobs 1 to 7, respectively, are 129, 48, 86, 135, 75, 48 and 235 units. The unconstrained completion times for machines 1 to 7 and 9 are 110, 80, 119, 92, 150, 30, 70 and 40 units respectively. The optimum solution is shown in the Gantt charts of Figure A5. The optimum completion times for jobs are 214, 65, 146, 244, 229, 48 and 245 units respectively. The optimum completion times for machines are 201, 183, 144, 244, 225, 220, 245 and 175 units.

## CONCLUSION

The developed GA is capable of solving more realistic cases of CJSS problems with returning tasks, machine availability limitation and job arrival time as well as the benchmark problems available in the literature. The new 2D chromosome presentation allows defining constraints and assessment criteria on machines as well as jobs.

## NOMENCLATURE

<i>C</i>	Completion time
<i>JA</i>	Job arrival time
<i>M</i>	Machine
<i>MA</i>	Machine available time
<i>N</i>	Number
<i>n</i>	Number
<i>S</i>	Start time
<i>T</i>	Process time
<i>t</i>	Task
<b>Subscript</b>	
<i>j</i>	Job
<i>m</i>	machine
<i>t</i>	task

## REFERENCES

- [1] Davis, L. (1985). Job shop scheduling with genetic algorithms. In Proceedings of an International Conference on Genetic Algorithms and Their Applications (Vol. 140). Carnegie-Mellon University Pittsburgh, PA.
- [2] Carlier, J., & Pinson, É. (1989). An algorithm for solving the job-shop problem. *Management science*, 35(2), 164-176.
- [3] Brucker, P., Jurisch, B., & Sievers, B. (1994). A branch and bound algorithm for the job-shop scheduling problem. *Discrete applied mathematics*, 49(1), 107-127.
- [4] Aarts, H., van Laarhoven, J., Lenstra, K., & Ulder, L. (1994). A computational study of local search algorithms for job shop scheduling. *ORSA Journal on Computing*, 6(2), 118-125.
- [5] Dorndorf, U., & Pesch, E. (1995). Evolution based learning in a job shop scheduling environment. *Computers & Operations Research*, 22(1), 25-40.
- [6] Mesghouni, K., Hammadi, S., and Borne, P. (1996). Production job-shop scheduling using genetic algorithms. In *Systems, Man, and Cybernetics. IEEE International Conference on* (Vol. 2, pp. 1519-1524). IEEE
- [7] Nowicki, E., & Smutnicki, C. (1996). A fast taboo search algorithm for the job shop problem. *Management science*, 42(6), 797-813.
- [8] Pinson, E. (1990). A practical use of Jackson's preemptive schedule for solving the job shop problem. *Annals of Operations Research*, 26(1-4), pp. 269-287.
- [9] Ying, W. and Bin, L. (1996). 'Job-shop scheduling using genetic algorithm', In *Systems, Man, and Cybernetics, IEEE International Conference on*, Vol. 3, pp. 1994-1999.
- [10] Williamson, P., Hall, A., Hoogeveen, A., Hurkens, J., Lenstra, K., Sevast'yanov, V., & Shmoys, B. (1997). Short shop schedules. *Operations Research*, 45(2), 288-294.
- [11] Lin, S. C., Goodman, E. D., & Punch III, W. F. (1997). A Genetic Algorithm Approach to Dynamic Job Shop Scheduling Problem. In *ICGA* (pp. 481-488).
- [12] Mesghouni, K., Hammadi, S., & Borne, P. (1998). On modelling genetic algorithms for flexible job-shop scheduling problems. *Stud. Inf. Control (Romania)*, 7(1), 37-47.
- [13] Vazquez, M., & Whitley, D. (2000). A comparison of genetic algorithms for the static job shop scheduling problem. In *Parallel Problem Solving from Nature PPSN VI* (pp. 303-312).
- [14] Garrido, A., Salido, A., Barber, F., & Lopez, A. (2000). Heuristic methods for solving job-shop scheduling problems. In *ECAI-2000 Workshop on New Results in Planning, Scheduling and Design*, Berlin (pp. 36-43).
- [15] Gonçalves, F., de Magalhães, J., & Resende, G. (2005). A hybrid genetic algorithm for the job shop scheduling problem. *European journal of operational research*, 167(1), 77-95.
- [16] Nowicki, E., & Smutnicki, C. (2005). An advanced taboo search algorithm for the job shop problem. *Journal of Scheduling*, 8(2), 145-159.
- [17] Zhang, Y., Li, P., Rao, Y., & Guan, Z. (2008). A very fast TS/SA algorithm for the job shop scheduling problem. *Computers & Operations Research*, 35(1), 282-294.
- [18] Hasan, S. K., Sarker, R., & Cornforth, D. (2008). GA with priority rules for solving job-shop scheduling problems. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence)*. IEEE Congress on (pp. 1913-1920). IEEE.
- [19] Li, Y., & Chen, Y. (2010). A genetic algorithm for job-shop scheduling. *Journal of software*, 5(3), 269-274.
- [20] Hao, X., Lin, L., Gen, M., and Ohno, K. (2013). Effective Estimation of Distribution Algorithm for Stochastic Job Shop Scheduling Problem. *Procedia Computer Science*, 20, 102-107.
- [21] Gutierrez, C. (2014). Overlap Algorithms in Flexible Job-shop Scheduling. *IJMAI*, 2(6), 41-47.
- [22] Peng, B., Lu, Z., & Cheng, T. C. E. (2015). A taboo search/path relinking algorithm to solve the job shop scheduling problem. *Computers & Operations Research*, 53, 154-164.

## APPENDIX A

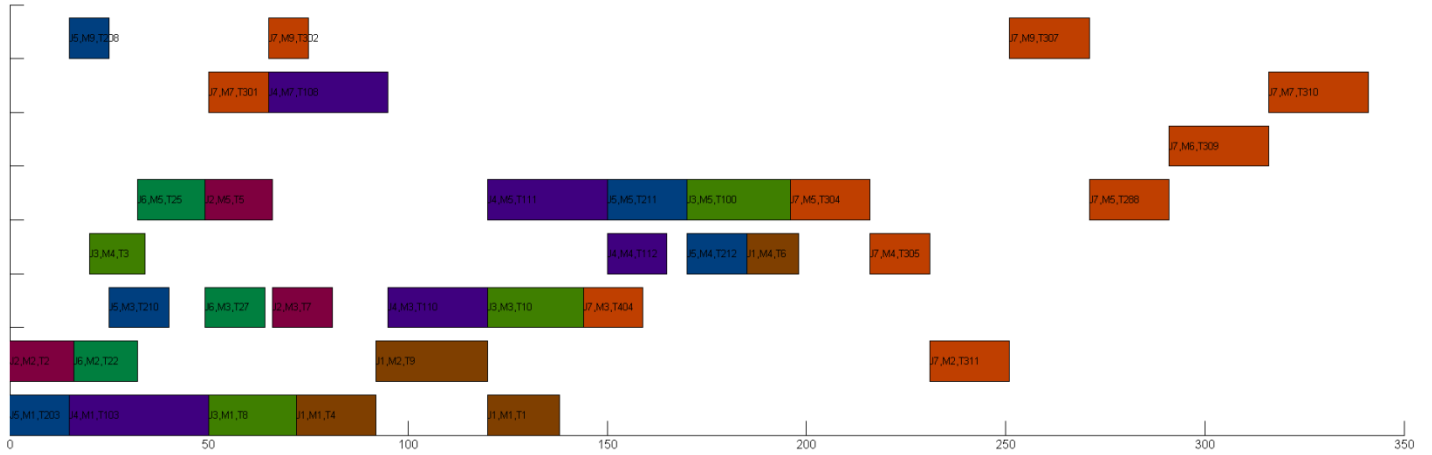


FIGURE A1-MACHINE GANTT CHART FOR CHROMOSOME OF FIGURE 2

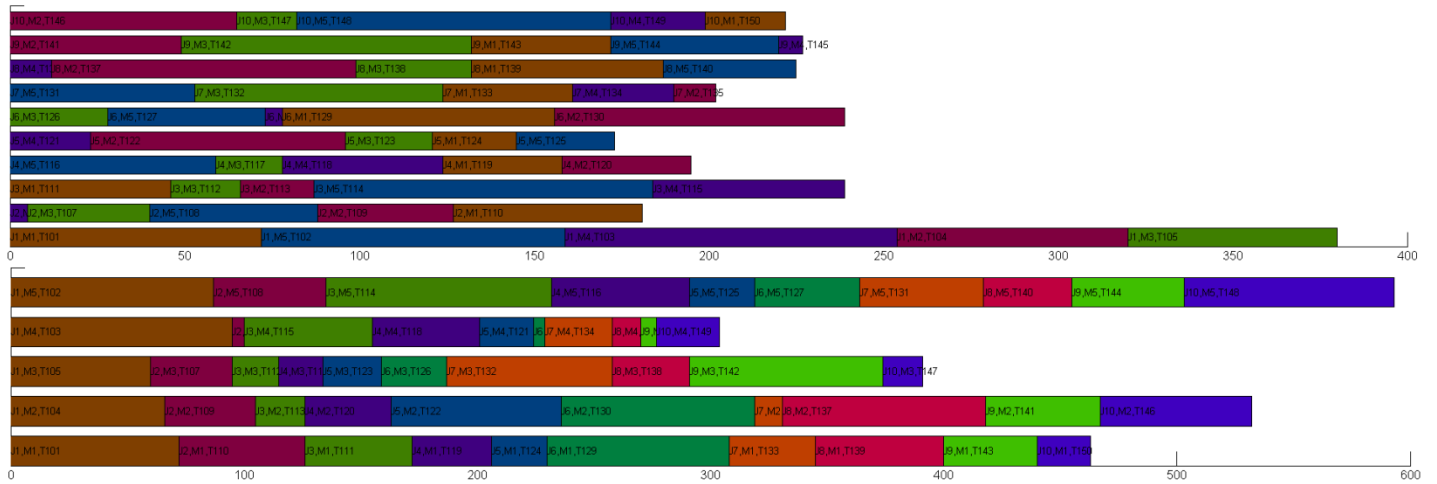


FIGURE A2-IDEAL JOB (TOP) AND MACHINE (BOTTOM) GANTT CHARTS AS THERE IS NO PRECEDENCE CONSTRAINTS FOR BENCHMARK PROBLEM LA05

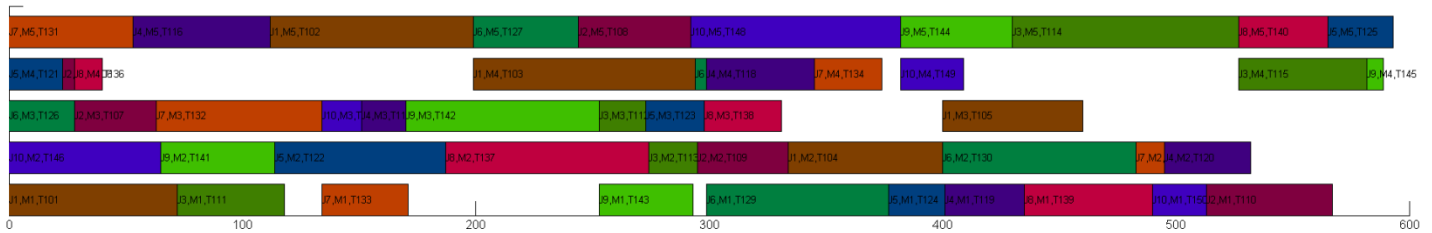
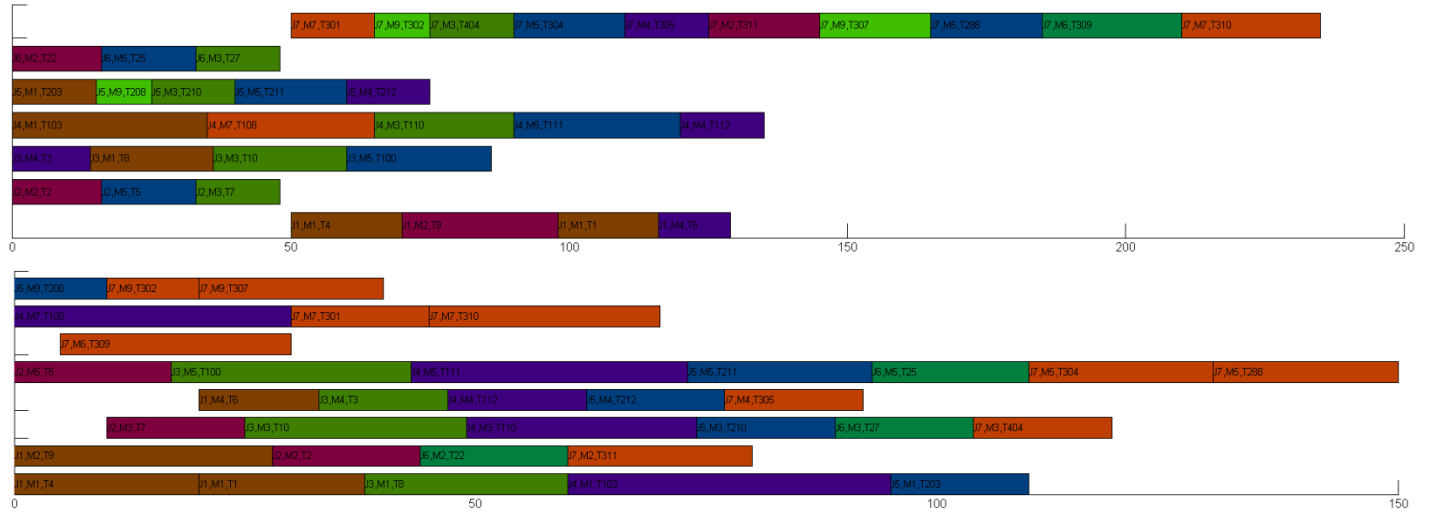
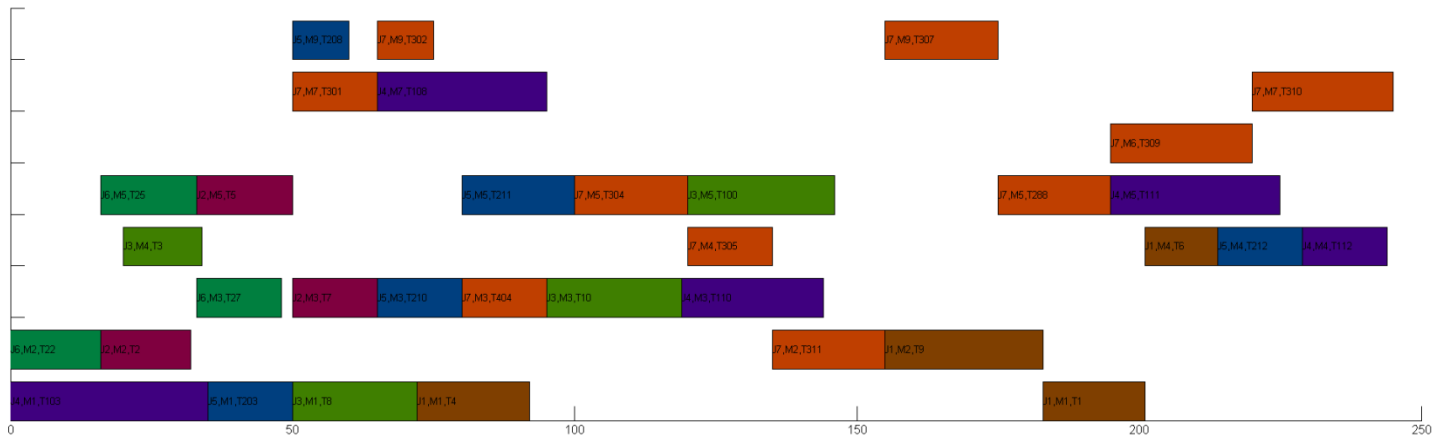


FIGURE A3-OPTIMUM MACHINE GANTT CHART FOR BENCHMARK PROBLEM LA05



**FIGURE A4-IDEAL JOB (TOP) AND MACHINE (BOTTOM) GANTT CHARTS AS THERE IS NO PRECEDENCE CONSTRAINTS FOR CJSS PROBLEM of FIGURE 1**



**FIGURE A5-OPTIMUM MACHINE GANTT CHART FOR CJSS PROBLEM of FIGURE 1**